You've got email ... again!
Protecting one's emailbox from spam with automatic filtering

Timo Salmi

Dedicated to Ilkka Virtanen on the occasion of his 60th birthday

Abstract

Salmi, Timo (2004). You've got email ... again! – Protecting one's emailbox from spam with automatic filtering. In *Contributions to Management Science, Mathematics and Modelling*. *Essays in Honour of Professor Ilkka Virtanen*. Acta Wasaensia No. 122, 225–244. Eds Matti Laaksonen and Seppo Pynnönen.

This paper outlines and exemplifies a spam (unsolicited commercial and other undesirable email) foiling system developed by the author. This automatic email filtering system is based on the concepts of whitelisting and blacklisting as adopted by the author in 1994. The paper suggests the idea of email password requirement returned automatically to the sender before accepting email as adopted by the author in 1997 for his own email protection. The paper also contributes the author's own method for testing the filtering recipes for procmail autonomous mail processor, and in an appendix a WWW-page based variant of the main password method.

"Looks like Timo had the right idea years ago!" (Berkes 2003). "For me, the day I read Professor Timo Salmi's webpage was a good day." (Clifford 2003).

Timo Salmi, Professor, Department of Accounting and Finance, University of Vaasa, P.O. Box 700, FIN–65101 Vaasa, Finland, e-mail ts@uwasa.fi, http://www.uwasa.fi/~ts/.

Acknowledgments

My thanks are due to Prof. Seppo Pynnönen for his comments in writing this paper. I also thank System Manager Hannu Hirvonen for many interesting computer related problem solving discussions over the years.

1. Introduction

It is well agreed that the main direction of the impact of computers on the then future society would have been very difficult to predict with any accuracy in the computers' early years in the 1950's. Originally regarded as mainly fast scientific calculators and commercial database processors, while also this aspect still is there, the computers' major domain at the beginning of the new millennium lies elsewhere. Their most profound impact has been in personal communication and services, and the consequent impact on society. This fact is based on two, originally unforeseen developments. The first was the personal computer (IBM 1981), considered just a curiosity at the outset. The second was the emergence of the Internet. Together these two have brought about a totally new era of global communication with all its benefits, but also its unfortunate, persistent abuses.

The paper at hand focuses on the most serious abuse and threat that has challenged the new flow of global communication almost since its outset (BBC News 2003, Templeton 2003 b). The phenomenon is Unsolicited Commercial Email, familiarly known as spam (Templeton 2003 a). With (according to some estimates) up to a half of the world-wide Internet bandwidth abused this way, spam poses a serious choking threat to the very texture of global communication. (Lemos 2002, Roberts 2002, Sounders 2002, Krim 2003, Paukku 2003, Siponen & Alatalo 2003). Even at best, spam and scams, such as the reinvented chain letter pyramid schemes (dmoz: Chain Letters, Osgoodby 2002, Watrous 2003) and the Nigerian Advance Fee Fraud (Arolainen 2003, US Secret Service), are huge nuisances for anyone with an email address on the Internet. Anyone with some experience on the Internet will have received such junk email to one's mailbox. A sobering indication of the extent of spam is a semi-random sample at the system-level at the University of Vaasa, Finland. On Monday, the 18th of August, 2003 the university's total email traffic amounted to 351713 messages. The collective filter at the university level intercepted 272656 messages (77.5% of the total email traffic) from 6564 different IP addresses as spam! And this figure still leaves unaccounted the spam that got through to the individual users. Such high figures are explained by the fact that the spammers systematically collect and generate target email lists. Quoting e.g. from the Federal Trade Commission (2002) spammers typically use computer programs that search public areas on the Internet to

harvest lists of email addresses extensively from web pages, newsgroups and chat rooms. (An even more comprehensive list of sources is listed e.g. by Raz.)

Another, often also unsolicited email related, but a much more sporadic modern threat is the spreading of the computer viruses/worms. This related, considerable threat would merit a consideration of its own. It is limited outside the scope of the current paper. However, the methods described for filtering one's email are directly applicable to the consequences of the email worms. (Such as e.g. Sobig.F which caused an unprecedented amount of bogus email traffic starting the 19th of August, 2003. See Appendix 3.)

This paper presents the ideas of an automatic spam filtering system, and exemplifies the methods on a UNIX-based email system. The concepts (in particular: whitelisting, blacklisting, and the email password requirement autoresponder) to be discussed can’t definitely be pinpointed to any single source of origin or a point of time. The present author does not claim proprietorship or strict originality on any of these concepts. Nevertheless, the methods, as presented in this paper, have been developed independently without known precedents by the current author. (Salmi 1997, 1999).

2. The outline of the automatic filtering design

When spam and other junk email pile in a person's mailbox at rates such as over 30 messages a day even at a collectively protected location it is advisable to consider resorting to automatic presorting of the incoming messages also at the individual user-level. Today many ISPs (Internet Service Providers) offer collective email protection already at the system level. The big problem in the collective method is finding the right balance between what is to be stopped as spam and what is to be let through. In the end it is the actual user who knows best. This fact speaks for the need of also individualized solutions.

Very generally speaking the idea is to direct the incoming email to different folders depending on where it comes from and what it contains. Such a system can grow very

complicated, and therefore a basic framework for thinking is needed. The method proposed here is made up of the following three layers (and then a considerable number of practical refinements).

1. Whitelisting: All email that appears on the database of whitelisted sources is let through. The whitelisted email can typically be further presorted to different folders (including one's main mailbox) according to e.g.
   - the sender,
   - the recipients (e.g. is it a mailing list),
   - the subject matter.
2. Blacklisting: All email that appears on the database of blacklisted sources (or subjects) is stopped. The blacklisted email can typically be further processed to
   - discard it outright ("/dev/null'ing"),
   - return it to the sender only, or
   - return it also to the sender's postmaster or abuse service, whilst
     - keep yourself a copy in a specified folder, or
     - keep yourself no copy.

3. Passwording: All the rest of the email is returned to sender with a password to be used in any further email, whilst
   - no copy is made, or
   - a copy made to a dedicated folder, while
   - additional, selective actions can be taken.

A note on terminology: On the Usenet news new words and meanings emerge with the new developments. (As a matter of fact "spam" is a good example of such a neologism.) In general terms methods where the sender is asked to take some action before getting through have become to be called C-R (challenge-response) systems or colloquially prove-you-love-me texts. Another interesting concept and method greylisting (Harris 2003) came to the fore in June 2003 in a thread in the news:comp.mail.misc Usenet newsgroup. It signifies measures where the delivery of a new, unidentified message is "falsely" bounced to require the common, automatic delayed resending. This may well be a good

idea, but compared to the password requirement the logic is a bit harder to follow if one is not well familiar with the details of the protocols which email programs use in handling bounces.

3. Getting started with the filtering

Given the general outline of the filtering system, to apply the delineated logic, the different cases have to be programmed as explicit rules. Some email programs, such as Microsoft Outlook Express (Microsoft Corporation) or Netscape Messenger (Netscape Marketing 2000) include the option for entering filtering rules. While useful, these options are somewhat limited in scope and in the range of actions that can be taken. The other possibility is to have a dedicated, separate filter program in front of the actual email program to do the filtering. Furthermore, complicated actions, once the incoming email is identified by the filter, can be carried out by various system tools.

In this paper the route of a separate filter program is taken. The platform for the examples to be presented will be a UNIX operating system. The email filtering program to be used in the examples is procmail ("autonomous mail processor"). Most of the actions will be programmed within procmail (procmail.org) as procmail recipes, but in more complicated cases the activities are piped to specific scripts using Bourne Shell scripting (Eriksson). Of course, in a way these are arbitrary choices, but they give a high flexibility of email filtering and they also are convenient for presenting the programming tasks involved. The downside of this flexibility will be that using such a system will require a fair amount of general understanding programming and a familiarity with the specific "power" tools chosen. However, they are much used by advanced users on the net community and they make a good demonstration of how to apply the outlined filtering design.

3.1. Setting up procmail

Provided that (as is usual) the email host system has the procmail program, the first step in installing the spam filtering system in a UNIX system is to redirect all incoming email to

the procmail program. The minute details and requirements of such setting up are better given in procmail manuals and e.g. advice WWW pages such as Salmi (1999) and its references. In a UNIX system email can be forwarded by creating a ~/.forward file with the following kind of contents

```
"|IFS=' ' && exec /usr/local/bin/procmail || exit 75 #myid"
```

The principle of piping all the incoming to the procmail is essential here, not so much the actual details of the arrangements and syntax, even if in actual practice they naturally must be entered correctly. In other words, this paper is not a program manual. The main purpose is to present the principles of spam foiling. Therefore, many minute details necessary for actually using such a procmail email filtering system, but not essential for the delineating the broad principles, are skipped and not commented here.

The crucial file in the procmail email filtering system is the ${HOME}/.procmailrc "procmail rcfile" configuration file which contains the "recipes" to process the incoming email. The recipes are made up of "rules". A simple case of .procmailrc is the following set, which blankly discards all "make money fast" email and puts all other incoming email into the user's mailbox.

```
:0                              # A new recipe starts with ":"
* ^Subject:.*Make money fast
/dev/null                       # This email is discarded

:0:                             # The second recipe starts
${DEFAULT}                      # The rest go to the user's mailbox
```

The rules of selection start with the "*" token. They use approximately the same syntax as UNIX egrep ("Search a file for a pattern using full regular expressions", see e.g. Stearns 1995) without an upper/lower case sensitivity. For example, were the email subject actually "how to make money fast **now**", the first recipe's rule would find a (regular expression) match and thus the first recipe's action would be taken.

3.2. Creating a procmail test instrument

As explained, the user enters the desired procmail recipes into his/her ${HOME}/.procmailrc (that is ~/.procmailrc) file to take different actions on the incoming email. For testing purposes this is not a good solution. The procmail recipes can grow fairly complicated, and furthermore email does not necessarily come in suitably at will. Therefore, a detached test method is needed to develop and evaluate individual procmail filtering recipes without disturbing one's regular email handling in the process. Full originality can hardly be claimed in creating such a testing system. Nevertheless, the one to be presented below has been independently put forward by the present author in Salmi (1999).

The testing system can be set up as follows. Create the following "proctest" file, preferably at the path. Make it executable using "chmod u+x proctest". Thus a new command "proctest" will always be available. The recipes to be tested are entered into the proctest.rc file, and the corresponding email to be tested is copied to the mail.msg file (e.g. if one uses UNIX elm "interactive mail system" or mutt "The Mutt Mail User Agent" the copying of an existing message can be done with the "C" copy command from within elm or mutt). Only one email message should be subjected to testing at a time to avoid confusing results.

```
#!/bin/sh
# The executable UNIX script file named "proctest".
# Bourne shell (the original UNIX shell /bin/sh) syntax is used.
#
# A test directory is needed.
# The exact location depends on the user's environment.
# The directory must be created if it does not already exist.
TESTDIR=/home/myid/test
#
# Feed an email message to procmail. Apply proctest.rc recipes file.
# First prepare a mail.msg email file to use for the testing.
procmail ${TESTDIR}/proctest.rc < ${TESTDIR}/mail.msg
#
# Display the outcome.
ls -lF ${TESTDIR}/Proctest.*
less ${TESTDIR}/Proctest.log
#
# Clean up.
rm -i ${TESTDIR}/Proctest.*
```

The procmail configuration file to be tested

```
# The proctest.rc test configuration file
#
# Setting some environment variables used by procmail
VERBOSE=yes
```

```
LOGABSTRACT=all
MAILDIR=${HOME}/test
LOGFILE=${HOME}/test/Proctest.log
#
# The recipes
:0:                           # The first recipe starts
* ^From:.*itv@.*uwasa\.fi
Proctest.itv                  # A folder for email from Ilkka Virtanen
#
:0:                           # The second recipe starts
Proctest.rest                 # All other email to this folder
```

An example email message copied to the mail.msg file

```
From itv@uwasa.fi Fri Jun 13 08:39:06 2003
Date: Fri, 13 Jun 2003 08:39:05 +0300
From: Ilkka Virtanen <itv@UWasa.Fi>
To: Timo Salmi <ts@uwasa.fi>
Subject: A new link

(The body of the message)

--
Ilkka Virtanen.
Professor of Operations Research and Management Science
Dean of the Faculty of Technology
University of Vaasa, POB 700, FIN-65101 Vaasa, Finland
Tel. 358-6-3248256, 358-50-5377909, Fax 358-6-3248557
E-mail: itv@uwasa.fi         http://www.uwasa.fi/~itv/
```

Captured output from the simple example test

```
poiju> proctest
procmail: [13340] Sat Jun 14 10:12:28 2003
procmail: Assigning "LOGABSTRACT=all"
procmail: Assigning "MAILDIR=/home/myid/test"
procmail: Assigning "LOGFILE=/home/myid/test/Proctest.log"
procmail: Opening "/home/myid/test/Proctest.log"
-rw-------   1 ts   ktt   958 Jun 14 10:12 /home/myid/test/Proctest.itv
-rw-------   1 ts   ktt   335 Jun 14 10:12 /home/myid/test/Proctest.log
procmail: Match on "^From:.*itv@.*uwasa\.fi"
procmail: Locking "Proctest.itv.lock"
procmail: Assigning "LASTFOLDER=Proctest.itv"
procmail: Opening "Proctest.itv"
procmail: Acquiring kernel-lock
procmail: Unlocking "Proctest.itv.lock"
From itv@uwasa.fi Fri Jun 13 08:39:06 2003
 Subject: A new link
  Folder: Proctest.itv                                                  479
rm: remove /home/myid/test/Proctest.itv (yes/no)? n
rm: remove /home/myid/test/Proctest.log (yes/no)? y
```

The email from Ilkka Virtanen used for the presented testing now resides in the Proctest.itv file and can be processed with any email program (elm is used in these tests). The essential tool to develop and test various situations at will is now readily available.

4. Whitelisting and blacklisting

4.1. A whitelist example

The first step in installing the spamfoiling system under observation is setting up a whitelist and a blacklist. Setting them up differs so little from each other that for the general idea it is sufficient to present setting up a whitelist and to observe that in blacklisting the email just would be discarded (to /dev/null) instead of directing it to a folder (Proctest.white) or (as would be more usual) directly to the user's email box (${DEFAULT}). The option of returning blacklisted email is taken up in the next section.

```
# The proctest.rc test configuration file
#
# Environment variables for procmail
VERBOSE=yes
LOGABSTRACT=all
MAILDIR=${HOME}/test
LOGFILE=${HOME}/test/Proctest.log
#
# Auxiliary definitions
# Get the sender's bare email address using formail mail reformatter program
# Ignore the Reply-To header-field:
FROMADDR=`formail -c -I"Reply-To:" -rt -xTo: \
         | expand | sed -e 's/^[ ]*//g' -e 's/[ ]*$//g'`
#
# Extract the sender's name from the first From: field
FROMNAME=`sed -e '/^$/ q' \
         | expand | egrep '^From: ' | head -1 \
         | sed -e 's/<//g' -e 's/>//g' \
         | sed -e 's/From: //' \
         | sed -e 's/^[ ]*//g' -e 's/[ ]*$//g'`
#
# The recipes
# Accept based on the potential appearance on either of the two whitelists
:0
* 1^0 $ ? echo "${FROMADDR}" | egrep -is -f /home/myid/test/whiteaddr.lst
* 1^0 $ ? echo "\"${FROMNAME}\"" | egrep -is -f /home/myid/test/whitename.lst
{
  :0
  { RULE="Accepted based on the generic whitelists" }
  :0:
  Proctest.white
}
#
:0:                             # The second recipe starts
Proctest.rest                   # All other email to this folder
```

Example contents of whiteaddr.lst

```
itv@.*uwasa\.fi
ts@.*uwasa\.fi
```

Example contents of whitename.lst

```
Ilkka.Virtanen
Timo.Salmi
```

It is easy to read in the example above the fact that setting up such a system is prohibitively complicated for an non-specialized user. The help of computer support personnel or some sort of preprocessed packages are needed for a more widespread

application of these ideas. This probably goes to explain why the kind of filtering described in this paper seems to be fairly rare despite the huge problem posed by the unrelenting spam-situation on the Internet. Fortunately, once the system is in place, it is reasonably easy for any user to write out the address lists such as whiteaddr.lst and whitename.lst.

Although the separate lists are convenient, the whitelisted and blacklisted names/ addresses could, of course, as well be inserted into the ~/.procmailrc configuration file. In fact, when filtering by the subject (see the "Make money fast" example in Section 3.1) all the parts would usually be contained within the configuration file, only.

4.2. Returning blacklisted email

As pointed out in Chapter 2, one of the options with blacklisted email is not just to discard it, but also to return it to the sender, or even make a copy to the sender's postmaster. Returning a message is of interest here because it demonstrates UNIX scripting (called by the "| piping") in conjunction with procmail. It is true that in actual practice much, if not most, of junk email on Internet comes from forged email addresses. Nevertheless, the example at hand shows the realization of the basic principle of returning a rejected message. An appropriate example recipe is given below:

```
# The recipes
# Safeguard against the possibility of email loops
:0:
* ^X-Loop:.*myid@myhost\.mydom
XLoop.mail

# Return blacklisted email
:0
* ^From:.*(\
BUSINESS.*TRAVEL.*LTD|\
MRS MURIYN JONAS SANIMBI|\
Vile.*Spammer)
{
  # First make a temporary file of the message to be returned
  :0c:formail.lock
  # Discard whitespaces from the said incoming email, insert a leading blank
  | expand | sed -e 's/[ ]*$//g' | sed -e 's/^/ /' > return.tmp
  #
  # Use formail -r mail reformatter program to resolve a suitable return address
  # Add a return subject and a loop safeguard to the outgoing email header
  # Construct and then send with sendmail program the rejection prepared
  #
  :0:formail.lock
  | (formail -r -I"Subject: Rejected mail: Recipient refusal" \
    -A"X-Loop: myid@myhost.mydom" ; \
    echo "--- begin rejected mail ---" ; \
    cat return.tmp ; \
```

```
    echo "--- end rejected mail ---" ; \
    rm -f return.tmp) \
    | /usr/lib/sendmail
}
```

In the above neither a separate blacklist file nor a separate script file is used. Instead all the rules and the actions are embedded into the recipe file for the current demonstration. In actual practice there are pros and cons to such a choice. All the handling stays in one file, which makes for a more concentrated documentation. On the other hand this way a real-life procmail configuration file tends to grow quite large and possibly quite convoluted.

Below is the current test rejection response that goes out. In actual practice the sender's address often would be masked and replaced by a non-returnable address to further guard against unwanted undeliverable email announcements and as a further safeguard against the potential email loops. Furthermore, an automatic copy could be sent to the blacklisted user's postmaster. However, those are practical details (sometimes a bit complicated), which do not add anything crucial to the ideas under observation here, and thus need not be demonstrated in the actual text. (Formulating the postmaster address is given in the Appendix 2. For more information on those aspects, see the procmail links in the references section of this paper, and the further links within those references.)

```
From ts@mail.uwasa.fi  Thu Jun 19 13:59:08 2003
Date: Thu, 19 Jun 2003 13:59:08 +0300 (EET DST)
From: <ts@mail.uwasa.fi>
To: ts@uwasa.fi
X-Loop: myid@myhost.mydom
Subject: Rejected mail: Recipient refusal

--- begin rejected mail ---
 From ts@uwasa.fi Thu Jun 19 12:59:14 2003
 Date: Fri, 13 Jun 2003 08:39:05 +0300
 From: Vile E. Spammer <forged@nowhere.com>
 To: ListOfSpamTargets <hidden@nowhere.com>
 Subject: Spam test
 Status: RO

 (The body of the test "spam" message)

    All the best, Timo;
    (posing as "Vile E. Spammer" for testing purposes)

 --
 Prof. Timo Salmi ftp & http://garbo.uwasa.fi/ archives 193.166.120.5
 Department of Accounting and Business Finance  ; University of Vaasa
 mailto:ts@uwasa.fi <http://www.uwasa.fi/~ts/>  ; FIN-65101,  Finland
 Timo's  FAQ  materials  at   http://www.uwasa.fi/~ts/http/tsfaq.html

--- end rejected mail ---
```

5. Requiring and identifying a password

The example in the previous section on returning blacklisted email to a great extent also gives the technical framework for sending out the password requirement. Only a few adjustments are needed. Thus part of the documentation comments can be omitted.

```
# Set your public email password
EMAILPASSW="abcd"

# Safeguard against the possibility of email loops
:0:
* ^X-Loop:.*myid@myhost\.mydom
XLoop.mail

# The whitelist and blacklist recipes here
#     ( T h o s e   r e c i p e s )

# Accept to your mailbox all email having the password on the subject line
:0:
* $ ^Subject:.*${EMAILPASSW}
${DEFAULT}

# For the rest of the incoming email, return the password requirement
:0
{
  SUBJ=`formail -xSubject: \
       | expand | sed -e 's/^[ ]*//g' -e 's/[ ]*$//g'`
  FROM=`formail -rt -xTo: \
       | expand | sed -e 's/^[ ]*//g' -e 's/[ ]*$//g'`
  :0:formail.lock
  | (formail -r -I"Subject: Returned email: Password required" \
    -A"X-Loop: myid@myhost.mydom" ; \
    echo "************************************************" ; \
    echo "* This is a computer-generated response message *" ; \
    echo "************************************************" ; \
    echo "" ; \
    echo "Dear ${FROM}" ; \
    echo "" ; \
    echo "Thank you for your email to me about" ; \
    echo "${SUBJ}" ; \
    echo "" ; \
    echo "To reach me, please include my public email password" ; \
    echo "${EMAILPASSW} anywhere on your subject line.") \
    | /usr/lib/sendmail
}
```

Using exactly the same test message as in the previous section the simplified requirement returned to the sender of the incoming email would look something like this:

```
From ts@mail.uwasa.fi  Thu Jun 19 23:31:40 2003
Date: Thu, 19 Jun 2003 23:31:40 +0300 (EET DST)
From: <ts@mail.uwasa.fi>
To: ts@uwasa.fi
X-Loop: myid@myhost.mydom
Subject: Returned email: Password required

************************************************
* This is a computer-generated response message *
************************************************

Dear forged@nowhere.com

Thank you for your email to me about
Spam test

To reach me, please include my public email password
abcd anywhere on your subject line.
```

In principle, that's all there is to the general outline of the password requirement method combined with whitelisting and blacklisting.

6. Conclusion

About five years of the author's practical experience with effectively the described password requirement method in place has turned out to be practically foolproof against unsolicited commercial email, i.e. the spam. (By a very rough approximation an order of 0.01% of the incoming spam has made it to the author's mailbox over the said period.) If one considers the layers of the logic outlined in Chapter 2, this is not at all surprising. If the sender has a forged address, as is common in spam, s/he'll never see the public password and thus will not get past the password requirement. In fact, s/he'll be unaware of its very existence. On the other hand, if s/he does get the password requirement response, it is extremely rare, even if not impossible, that the spammer would actually use the required public password. Why? As stated in Salmi (1997) "By its very nature spamming is a huge mass activity. There is no way the spammers have the resources to customize in order to bypass a single individual's public password protection. Furthermore, contrary to a getting a new, spam-free user id, an email password is easy to change anytime."

The principles described in this paper to solve the prohibitive junk email problem are straightforward and simple. Unfortunately, programming and setting them up is obviously far too complicated for an lay-person PC user to do unaided. "Easy to describe, complicated to install." Therefore, practical methods to set up the ideas presented are needed. The incentives to do so fall into the realm of commercial programming. In fact, at the time of writing this, there are some movements in evidence on the Internet towards this direction. One of such examples is Spam Arrest (2003) which is based on directing the email sender to an identification requirement acted out on a dedicated web page. That idea has much in common with the variant presented in Appendix 1. Another similar, recent example is Spam Catcher (2003). Another consequence of the complicated nature of

filtering is that quality ISPs increasingly will have to offer improved, collective spam protection already at the system level.

There also is another dilemma involved with the password requirement that has to be observed. The downside of such a junk email prevention method is that since it is very effective it also will curb some legitimate email contacts. This is not a prohibitive problem, when whitelisting is a practical proposition, but for example commercial firms with huge potential customer bases just can't afford losing the potential contacts because of a password requirement that some users will find excessive or even offensive. The system is much better fitted for a private user or a user where the contacts typically remain e.g. within, say, a university environment circle.

A third obvious dilemma definitely worth further consideration is the logic of initiating a contact between two users of the email password system when the parties are previously unknown to each other (see however, Appendix 1). Because the password system still is rather rare this problem has not yet risen more to the fore. Some common protocol is probably in order just like in the exchange of public keys in PGP exchanging encrypted messages (see e.g. Singh 1999). The options include publicizing one's public email password on one's web page, in one's email signature, or accepting email sent through one's webpage (since then special arrangements are easy to set up). In fact, the author utilizes such measures.

Be the problems with the presented method as may, it is unavoidable that one way or the other the junk email problem will eventually have to be solved before it manages to choke the usefulness of the global Internet. It only is to be hoped that the ideas presented in this paper and on the net by the present author might make up one modest step towards a resolution.

References

Arolainen, Teuvo (2003). Nigerialaiskirjeiden tulva jatkuu. *Helsingin Sanomat* 5.5.2003, A7.

Berkes, Jem (2003). *Subject: Re: anti spam* [online] [cited 28-May-2003]. A Usenet news posting in the newsgroup news:comp.mail.misc. Date: Tue, 27 May 2003 15:59:05 GMT. Message-ID: Xns93886FCFED449jbuserspc9org@205.200.16.73

BBC News [online] (2003). Spam celebrates silver jubilee. 4-May-2003 [cited 6-May-2003]. Available from Internet: http://news.bbc.co.uk/2/hi/technology/2996319.stm

Clifford, Alan (2003). *Subject: Re: The greylisting idea, effective?* [online] [cited 25-June-2003]. A Usenet news posting in the newsgroup news:comp.mail.misc. Date: Wed Jun 25 01:38:17 EET DST 2003. Message-ID: Pine.LNX.4.53.0306242249540.19119@ mundungus.clifford.ac

dmoz [The Open Directory Project] [online]. Logical search path: Top: Society: Issues: Fraud: Internet: Make Money Fast: Chain Letters [cited 14-May-2003]. Available from Internet: http://dmoz.org/Society/Issues/Fraud/Internet/Make_Money_Fast/Chain_Letters/

Eriksson, Era. *An Introduction to the Unix Shell; An HTMLized version of Steve Bourne's original shell tutorial*. [online] [cited 17-May-2003]. Available from Internet: http://rhols66.adsl.netsonic.fi/era/unix/shell.html

Federal Trade Commission (2002). Consumer Alert. *Email Address Harvesting: How Spammers Reap What You Sow*. [online] [cited 3-September-2003]. Available from Internet: http://www.ftc.gov/bcp/conline/pubs/alerts/spamalrt.pdf

Harris, Evan (2003). *The Next Step in the Spam Control War: Greylisting* [cited 24-June-2003]. Available from Internet: http://projects.puremagic.com/greylisting/

IBM (1981). Personal computer announced by IBM. *IBM (Information Systems Division, Entry Systems Business) Press Release* August 12, 1981 [online] [cited 23-June-2003]. Available from Internet: http://www-1.ibm.com/ibm/history/documents/pdf/pcpress.pdf

Krim, Jonathan (2003). Spam's Cost To Business Escalates: Bulk E-Mail Threatens Communication Arteries. *Washington Post* Thursday, March 13, 2003; Page A01 [online] [cited 5-May-2003]. Available from Internet: http://www.washingtonpost.com/wp-dyn/ articles/A17754-2003Mar12.html

Lemos, Robert (2002). Spam hits 36 percent of e-mail traffic. *CNET News.com* August 29, 2002, 4:48 AM PT [online] [cited 5-May-2003]. Available from Internet: http://zdnet.com.com/2100-1106-955842.html

Microsoft Corporation. *Home Page for Microsoft Outlook.* [online] [cited 17-May-2003]. Available from Internet: http://www.microsoft.com/office/outlook/

Netscape Marketing (2000). *Netscape Messenger*. [online] [cited 17-May-2003]. Available from Internet: http://wp.netscape.com/communicator/messenger/v4.0/index.html

Osgoodby, Bob (2002). What Is A Ponzie? *INSIDER REPORT* Monday, October 07, 2002 [online] [cited 14-May-2003]. Available from Internet: http://www.insiderreports.com/ storypage.asp_Q_ ChanID_E_MR_A_StoryID_E_20001095

Paukku, Timo (2003). Roska@joukko tukkii sähköpostit. *Helsingin Sanomat* 5.7.2003, C15.

Raz, Uri. How do spammers harvest email addresses? [online] [cited 4-September-2003]. Available from http://www.private.org.il/harvest.html

procmail.org *The home page of the procmail mail processing and SmartList mailing list suites*. [online] [cited 17-May-2003]. Available from Internet: http://www.procmail.org/

Roberts, Paul (2002). Holidays Bring a Whole Lot of Spam. *PCWorld.com* Monday, December 23, 2002 [online] [cited 5-May-2003]. Available from Internet: http://www.pcworld.com/news/article/0,aid,108174,00.asp

Salmi, Timo (1997). *Foiling Spam with an Email Password System*. [online] [cited 5-May-2003]. Available from Internet: http://www.uwasa.fi/~ts/info/spamfoil.html

Salmi, Timo (1999). *Timo's procmail tips and recipes*. [online] [cited 5-May-2003]. Available from Internet: http://www.uwasa.fi/~ts/info/proctips.html

Saunders, Christopher (2002) Study: E-mail to Double by 2006. *AtNewYork.com* [online] [cited 5-May-2003]. Available from Internet: http://www.atnewyork.com/news/article.php/1471801

Singh, Simon (1999). *Code Book. The Secret History of Codes & Code-breaking*. Fourth Estate, London.

Siponen, Mikko T. & Toni Alatalo (2003). Roskapostilta voi suojautua. Vieraskynä, *Helsingin Sanomat* 20.6.2003, A5.

Spam Arrest (2003). [online] [cited 24-June-2003]. Available from Internet: http://spamarrest.com/

Spam Catcher (2003). [online] [cited 16-August-2003]. Available from Internet: http://www.softouch.on.ca/spatcher/

Stearns, Bob (1995). UNIX regular expressions. *The UCNS Computer Review* Spring Quarter [online] [cited 23-June-2003]. Available from Internet: http://www.uga.edu/ ~ucns/tti/Computer_Review/Spring95/Regular_expressions.html

Templeton, Brad (2003 a). Reflections on the 25th Anniversary of Spam. [online] [cited 5-May-2003]. Available from Internet: http://www.templetons.com/brad/spam/spam25.html

Templeton, Brad (2003 b). Origin of the term "spam" to mean net abuse. [online] [cited 5-May-2003]. Available from Internet: http://www.templetons.com/brad/spamterm.html

United States Secret Service. *[Nigerian] Advance Fee Fraud Advisory*. [online] [cited 5-May-2003]. Available from Internet: http://www.secretservice.gov/alert419.shtml

Watrous, Donald (2003). *Chain letters* [cited 14-May-2003]. Available from Internet: http://www.cs.rutgers.edu/~watrous/chain-letters.html

Appendix 1: The WWW-page variant of spam avoidance

There is a fairly simple variation to complement (rather than replace) the public email password system. One could call it the WWW-page variant or even the Guestbook rendition. It goes as follows: in building a WWW page the HTML code for sending email via the WWW page is

```
<A HREF="mailto:myid@myhost.mydom">Click to send me email</A>
```

If one changes it to

```
<A HREF="mailto:myid@myhost.mydom(FirstName LastName Passwd)">Click to send me email</A>
```

Then the procmail recipe for accepting such email simply is e.g.

```
:0:
* ^To:.*FirstName.*LastName.*Passwd
WWW.mail
```

Naturally, this method is not a guarantee against spam. But in the author's experience it can be sufficiently effective in actual practice. The author has had this complementary variation on the side of the main email password spam foiling method on several web pages since 1998. Almost no additional spam has resulted via this route. A big advantage of this method is that it at least partly avoids the first contact conundrum if both the parties are using a challenge-response email password system.

The same contribution situation goes of the WWW-page variant as for the principal password-based challenge-response spam foiling method presented in this paper. The present author does not claim an actual originality of the idea. Nevertheless, the method presented in this appendix has been developed independently without known precedents by the current author.

Appendix 2: An advanced example recipe for detecting Korean email

Much spam seems to originate from Korea. Detecting if the incoming email is in Korean (or Chinese or Cyrillic) is given below as one example of a complicated task and the corresponding heuristic recipe made possible in procmail filtering. The original idea owes to a WWW page (no longer available at the original address) by Walter Dnes. The demonstration below also includes getting the sender's postmaster's address. The example is an extract from Salmi (1999).

```
# Get the sender's address, ignore Reply-To:
FROM_=`formail -c -I"Reply-To:" -rt -xTo: \
  | expand | sed -e 's/^[ ]*//g' -e 's/[ ]*$//g'`

# Get the sender's host
FHOST_=`echo "${FROM_}" | awk -F@ '{ print $2 }'`

# Your path to sendmail
SENDMAIL="/usr/lib/sendmail"

# Reject probable Korean, Chinese or Cyrillic email using character scoring
:0
* ! ^X-Loop:.*myid@myhost\.mydom
* ! $ ? echo ${FHOST_} | fgrep -is 'myhost.mydom'
* $ ? echo ${FHOST_} | fgrep -is '.'
{
  :0BD
  *  -1^1 .
  *   2^1 =[0-9A-F][0-9A-F]
  *  20^1 [¡¢£€¥_§¨©ª«¬®¯°±¸¹´µ¶·¸¹º»¼½¾¿]
  *  20^1 [ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ_ÑÒÓÔÕÖ_ØÙÚÛÜ__ß]
  *  20^1 [àáâãäåæçèéêëìíîï_ñòóôõö÷øùúûü__ÿ]
  *  20^1 =[89A-F][0-9A-F]
  * -20^1 [ÅÄÖåäçèéêë]
  * -20^1 =(E5|C5|E4|C4|F6|D6|E0|E1|E2|E7|E8|E9|EA|EB)
  {
    :0
    { RULE="Probable Korean email" }
    #
    :0c:${HOME}/procmail.lock
    | expand | sed -e 's/[ ]*$//g' \
      | sed -e 's/^/ /' > ${HOME}/procmail.reject.korean
    #
    :0:${HOME}/procmail.lock
    | (formail -r -I"Subject: Autorejected email" \
      -I"To: ${FROM_}" \
      -I"Cc: postmaster@${FHOST_}" \
      -A"X-Loop: myid@myhost.mydom" ; \
      echo "--- begin rejected probable Korean email ---" ; \
      echo "" ; \
      cat ${HOME}/procmail.reject.korean ; \
      echo "--- end of rejected probable Korean email ---" ; \
      rm -f ${HOME}/procmail.reject.korean) \
        | ${SENDMAIL}
  }
}
```

Appendix 3: Identifying a virus/worm in the email

Consider as an example avoiding the Sobig.F virus/worm which is spread by email. In terms of the techniques presented in the current paper, Sobig.F email is blacklisted. This virus and its consequences caused a then unprecedented load on the Internet in the last third of August 2003. In a way such email is easier to detect and avoid than the ordinary spam, since there is a high regularity to the pattern. In Sobig.F email the subject always is one of a limited selection, there always is an unvarying "X-MailScanner:" header, and the body of the message always contains text from just two alternatives. These facts enable a simple, efficient detection recipe with a reasonably low probability of false positives. The example is from the author's own procmail configuration file. The "RULE" variable in the recipe is for helping to separately record which of the many recipes in the configuration file has been enacted.

```
# Discard Sobig.F
# Look at the headers
:0H
* ^Subject.*(\
Re: Thank you\!|\
Thank you\!|\
Your details|\
Re: Details|\
Re: Re: My details|\
Re: Approved|\
Re: Your application|\
Re: Wicked screensaver|\
Re: That movie)
* ^X-MailScanner: Found to be clean
{
  # Look at the message body
  :0B
  * (Please)? see the attached file for details
  {
    :0
    { RULE="Sobig.F" }
    :0
    /dev/null
  }
}
```